Theoretical Computer Science 107 (1993) 77–94
Elsevier

Unambiguity of circuits

Klaus-Jörn Lange*

Institut für Informatik, Technische Universität München, Arcisstr. 21, D-8000 München 2, Germany

Abstract

Lange, K.-J., Unambiguity of circuits, Theoretical Computer Science 107 (1993) 77–94.

The concept of unambiguity of circuits is considered. Several classes of unambiguous circuit families within the NC-hierarchy are introduced and related to unambiguous automata and to PRAMs with exclusive write access. In particular, we show CREW-TIME($\log^k n$) = UnambAC^k for each positive integer k.

1. Introduction

The central object of interest of parallel complexity theory is the class NC (see e.g. [7]), which can be characterized by several devices, among them being parallel random-access machines [9, 10], boolean circuits [3], alternating Turing machines [5], and deterministic and nondeterministic auxiliary pushdown automata [6].

With appropriate complexity bounds these devices recognize exactly the languages in NC. If we take a closer look at the time or depth bounds, by considering functions of order $O(\log^k n)$ for any fixed k instead of $O(\log^{O(1)} n)$ we get several intertwining hierarchies, the union of which is NC. Located between DSPACE($\log n$) and P, these two classes would be separated if these hierarchies could be shown to be proper, i.e. if it could be shown that the increase of time or depth from $O(\log^k n)$ to $O(\log^{k+1} n)$ leads to different classes for some k. Some of these hierarchies coincide, thus characterizing the relation between different models of parallel computations. In particular, the following equivalences have been obtained:

- time of a CRCW-PRAM, depth of alternation on a Turing machine, and depth of a uniform circuit of unbounded fan-in [21],
- time of a nondeterministic auxiliary pushdown automaton, treesize of an alternating Turing machine, and depth of a uniform circuit of semi-unbounded fan-in [18, 24], and

*This work was supported by the Deutsche Forschungsgemeinschaft, SFB 342: TP A4, and by the International Computer Science Institute.

- time of an alternating Turing machine and depth of a uniform circuit of bounded fan-in [19].

In addition, on a less general level, Dymond and Ruzzo [8] showed a similar relation between the time of a CROW-PRAM and the time of a deterministic auxiliary pushdown automaton. As we see, this list contains neither CREW- nor EREW-PRAMs, which are not characterized by any other parallel or sequential device, albeit most PRAM algorithms are working on PRAMs of these types. Our idea is to break this isolation by pursuing the following relationship between context-free languages and PRAM-types:

- $CFL \subseteq CRCW\text{-}TIME(\log n)$ [18],
- $UnambCFL \subseteq CREW\text{-}TIME(\log n)$ [20],
- $DCFL \subseteq CROW\text{-}TIME(\log n)$ [8].

It is the aim of this paper to further relate the concepts of exclusiveness and of unambiguity. This we want to do by considering the concept of unambiguity of circuits. Here we have to distinguish the notions of unambiguity and of uniqueness. While the first one uses uniqueness of an acceptance path as a restriction of computational power, the second one increases computational power by using uniqueness of an acceptance path as a tool. That is, in the first case we forbid the existence of multiple acceptance paths while in the second case we might use this fact to reject a computation. Thus, the concept of uniqueness applied to circuits would lead us to use something like unbounded fan-in gates for unique existence. In contrast to this, an unambiguous circuit has to restrict ways of multiple acceptance. We do this in the following way: Just as in a CREW- or EREW-PRAM we assume the machine and the algorithm to avoid any multiple access to a memory cell, and may think that in case of violation of this rule the corresponding cell might be destroyed; and assume the circuit to avoid any multiple 1-input to OR-gates of unbounded fan-in (multiple 0-input to AND-gates of unbounded fan-in). Otherwise, that is if an OR-gate of unbounded fan-in gets two or more 1's as input, we have no information of the output of this gate. Unambiguous circuit classes defined in this way show very close relationships not only to CREW-PRAMs but also to unambiguous Turing machines and auxiliary pushdown automata [14, 26].

In particular, we characterize CREW-PRAM's in terms of unambiguous circuits by the equation $CREW\text{-}TIME(\log^k n) = UnambAC^k$. This answers an open question posed in [21].

2. Preliminaries

The reader is assumed to be familiar with the basic concepts and notations in formal language and computational complexity theories as they are contained for instance in [11] or [25]. In addition, we often use the notion $LOG(\mathscr{A})$ to denote the class of all languages reducible by logspace many–one reductions to members of the

class $\mathcal{A}$. If $X(f(n))$ denotes a complexity class with a resource bound $f(n)$ we let $X(\text{pol})$ be the union of all $X(p(n))$ over all polynomials $p(n)$.

In the following we are going to review briefly some concepts and facts of parallel complexity theory. Detailed definitions and constructions may be found in [7, 1, 2, 16] or in the cited references.

2.1. Circuits

In this section we present some basic facts concerning boolean circuits. A *circuit* C is a finite acyclic graph. Nodes of indegree (outdegree) zero are called *inputs* (*outputs*). Inner nodes are labelled by boolean functions, throughout this paper by negations, disjunctions, and conjunctions. The inner nodes are often called *gates* and the edges *wires*. Given an assignment of boolean values to all inputs, each gate evaluates to either TRUE (1) or FALSE (0) according to the interconnection structure of C. If C has just one output, we might use C to recognize binary languages defining $L(C)$ to be the set of assignments to the inputs which let the output evaluate to TRUE. The *size* of C is the number of its gates (i.e. not counting the inputs).[1] The *depth* of C is the length of the longest path connecting an input with an output.

A circuit family $\mathcal{C}$ is a set $\{C_n \mid n \geq 0\}$ of circuits, where each C_n has exactly n inputs. $\mathcal{C}$ has *polynomial size* iff for some polynomial $p(\cdot)$, the size of each C_n is bounded by $p(n)$. Similarly, the *depth* of $\mathcal{C}$ is *bounded by* $\log^k n$ iff for some constant $c > 0$ the depth of each C_n is bounded by $c \log^k n$. Furthermore, $\mathcal{C}$ is of *bounded fan-in* iff for some positive integer m (usually 2) the indegree of each gate in each C_n is bounded by m. $\mathcal{C}$ is of *semi-unbounded fan-in* iff no C_n contains a negation[2] and for some positive integer m in each C_n the indegree of each gate labelled as a conjunction is bounded by m. If there is no bound on the indegrees we say that $\mathcal{C}$ is of *unbounded fan-in*. When working with unambiguous circuits we distinguish between gates of bounded and of unbounded fan-in, since these will have to be treated differently. Thus, when necessary, we will denote OR- and AND-gates of unbounded fan-in as $\exists$- and $\forall$-gates. This means that with $\mathcal{C}$ there exists a positive integer m such that in each C_n each OR- and each AND-gate has a fan-in bounded by m, while each $\exists$- and each $\forall$-gate within some C_n may have a fan-in as large as the size of C_n.

In order to relate classes of languages defined by circuits with complexity classes, it is necessary to consider *uniform* circuit families by requiring that the members of a circuit family are "sufficiently similar" to each other. There are several uniformity conditions which fortunately turned out to be equivalent in most cases [19]. Throughout this paper we use the notion of DSPACE($\log n$)-uniformity: $\mathcal{C}$ is called DSPACE($\log n$)-*uniform* iff the mapping $n \mapsto \langle C_n \rangle$ is computable within logarithmic

[1] Sometimes the number of wires is a more appropriate measure. But since we work with circuits of polynomial and not of polylogarithmic size, this would make no difference.

[2] To make this concept reasonable we have to assume that all inputs are given together with their negations, i.e. each C_n has $2n$ inputs.

space. Here $\langle C_n \rangle$ denotes an adequate coding of a circuit. We call the logspace machine computing this mapping the *uniformity machine.*

For $k \geqslant 1$ we denote by NC^k (SAC^k, AC^k) the families of languages recognizable by DSPACE(log n)-uniform, polynomially sized, $O(\log^k n)$-depth-bounded circuit families of bounded (semi-unbounded, unbounded) fan-in.[3] The following relations are well-known:

$$NC^k \subseteq SAC^k \subseteq AC^k \subseteq NC^{k+1}$$

and

$$NC^1 \subseteq DSPACE(\log n) \subseteq NSPACE(\log n) \subseteq SAC^1 \subseteq AC^1.$$

It has to be mentioned that throughout this paper only *layered* circuits are considered. That is, we assume that for every gate a in a circuit C there exists a *height* $h(a)$ such that every directed path from any input to a has length $h(a)$; in particular, each predecessor of a has height $h(a)-1$. (Some authors demand that in a *layered* circuit existential and universal gates alternate. In addition, it seems reasonable to include the height of gates in the description of a layered circuit, i.e. to let the height function in a uniform circuit family be computable by the uniformity machine.) This is no restriction for NC, SAC, or AC-circuits. But we were unable to show a corresponding "normal form" result for the case of unambiguous AC-circuits, introduced in Section 3. On the other hand, all constructions of unambiguous circuits in this paper are of a levelled structure, in which each level is a circuit of bounded depth and its outputs are given as inputs either to the next level or to the final one. A circuit of this structure can be transformed easily into a layered circuit.

2.2. *Parallel random-access machines*

The concept of a PRAM goes back to [9, 10]. Roughly, a PRAM is a set of random-access machines, called *processors*, working synchronously and communicating via a *global memory*. Each step takes one time unit, regardless of whether it performs a local or a global (i.e. remote) operation. All processors execute in parallel the same sequence of statements $S_1, S_2, \ldots, S_K$ which is independent of the input. Let each processor have *local memory* cells $L_1, L_2, \ldots, L_{q(n)}$, and let $G_1, G_2, \ldots, G_{q(n)}$ be the cells of global memory, where n is the length of the input, which is given in $G_1, \ldots, G_n$ and q is a polynomial. A computation has ended if all processors have reached a HALT statement. Each statement S_μ is of one of the following types:

Indirect Writes:

$$G_{L_a} = L_b \quad \text{and} \quad L_{L_a} = L_b,$$

Indirect Reads:

$$L_a = G_{L_b} \quad \text{and} \quad L_a = L_{L_b},$$

[3] Actually, a reasonable definition of NC^1 seems to require a uniformity notion more delicate than DSPACE(log n)-uniformity, yielding the probably smaller class ATIME(log n) [19].

Binary Operation:

$L_a = L_b \circ L_c$,[4]

Constants:

$L_a = \langle \text{constant} \rangle$, $L_a = \text{LENGTH}$,[5] or $L_a = \text{PIN}$,[6]

Jumps:

goto S_a or **if** $L_a > 0$ **then goto** S_b,

Others:

HALT or NOOP.

Throughout this paper we consider PRAMs where both the number of processors and all occurring data and addresses are bounded polynomially in the length of the input; the latter condition is equivalent to a logarithmic bound on the word length of all global and local memory cells.

There are several versions of this device concerning the way in which the simultaneous memory access is handled. There are three versions concerning the write access. A machine with *concurrent write* access allows the simultaneous write access of several processors to the same memory cell in one step. There are several conventions as to how to solve this conflict, i.e. how to determine which will be the new value of the referenced cell. Fortunately, in our context all these methods are equivalent. A machine with *exclusive write* access forbids simultaneous writes and requires that in each step at most one processor may change the content of a global memory cell. A machine with *owner write* access is even more restricted by assigning to each cell of global memory a processor, called the *write-owner*, which is the only one to have write access to this memory cell. Correspondingly, we get three ways to manage read access to the global memory: *concurrent read*, *exclusive read*, and *owner read*. In this way we get nine versions of PRAMs, denoted as $X\text{R}Y\text{W}$-PRAMs with $X, Y \in \{\text{O}, \text{E}, \text{C}\}$, where $X\text{R}$ specifies the type of read access and $Y\text{W}$ that of the write access, where the access types are designated by their initials. By $X\text{R}Y\text{W}\text{-TIME}(f(n))$ we denote the class of all languages recognizable in time f by $X\text{R}Y\text{W}$-PRAMs with a polynomial number of processors. By definition we know that $X\text{R}Y\text{W}\text{-TIME}(f) \subseteq X'\text{R}Y'\text{W}\text{-TIME}(f)$ for $X, X', Y, Y' \in \{\text{O}, \text{E}, \text{C}\}$ if $X \leqslant X'$ and $Y \leqslant Y'$, where we set $\text{O} \leqslant \text{E} \leqslant \text{C}$.

By [21, 17] we have the following relationships for $k \geqslant 1$:

$$\text{CRCW-TIME}(\log^k n) = \text{AC}^k,$$

$$\text{NC}^k \subseteq \text{OROW-TIME}(\log^k n),$$

$$\text{DSPACE}(\log n) \subseteq \text{OROW-TIME}(\log n),$$

[4] Since the word length is logarithmically bounded and since we deal with DSPACE($\log n$)-uniform circuits, we can admit arbitrary DSPACE($\log n$)-computable functions.

[5] This gives length of the input, i.e. n.

[6] This gives the *processor identification number*.

$$\text{ORCW-TIME}(f)=\text{ERCW-TIME}(f),$$

$$\text{OREW-TIME}(f)=\text{EREW-TIME}(f).$$

Remark 2.1. In the most powerful model, the CRCW-PRAM, the global memory behaves like a shared memory, since each processor can access each cell of global memory. However, in the most restricted model, the OROW-PRAM, the global memory is deteriorated to a set of one-directional channels between pairs of processors. Thus, an OROW-PRAM is something like a completely connected synchronous network. Although this model seems to be much more restricted, the relation

$$\text{NC}^k \subseteq \text{OROW-TIME}(\log^k n) \subseteq \text{CRCW-TIME}(\log^k n) = \text{AC}^k \subseteq \text{NC}^{k+1}$$

indicates that it is a model as "parallel" as a CRCW-PRAM.

2.3. *Auxiliary pushdown automata*

Another device that is interesting in this context is the *auxiliary pushdown automaton* which might be thought of as a pushdown automaton augmented with an $S(n)$-space-bounded working tape and a two-way access to its input. Cook [6] showed that both the deterministic and the nondeterministic versions of this automaton type recognize exactly the class $\bigcup_{c>1} \text{DTIME}(c^{S(n)})$ of languages acceptable in deterministic time exponential in $S(n)$. Throughout this paper we work with the case $S(n)=\lceil \log n \rceil$.

By restricting this device with an additional time bound we get classes down in the NC-hierarchy. Let $\text{DAuxPDA-TIME}(f(n))$ be the class of all languages recognizable by deterministic auxiliary pushdown automata with logarithmically space-bounded working tapes which are time-bounded by $O(f(n))$. $\text{NAuxPDA-TIME}(f(n))$ denotes the corresponding nondeterministic class. By [24, 19, 8] we have the following relationships for $k \geq 1$:

$$\text{DAuxPDA-TIME}(\text{pol})=\text{CROW-TIME}(\log n),$$

$$\text{NAuxPDA-TIME}(c^{\log^k n})=\text{SAC}^k,$$

$$\text{NC}^k \subseteq \text{DAuxPDA-TIME}(c^{\log^k n}).$$

Furthermore, Sudborough [23] showed that

$$\text{DAuxPDA-TIME}(\text{pol})=\text{LOG}(\text{DCFL}) \quad \text{and}$$

$$\text{NAuxPDA-TIME}(\text{pol})=\text{LOG}(\text{CFL}).$$

Finally, we mention in passing that many classes defined by circuits, PRAMs, or AuxPDAs possess characterizations in terms of alternation. Thus, $\text{CRCW-TIME}(\log^k n)=\text{AC}^k$ corresponds to depth of alternation, $\text{NAuxPDA-TIME}(c^{\log^k n})=\text{SAC}^k$ corresponds to treesize-bounded alternation, and NC^k corresponds to alternating time [18, 19, 21, 24].

3. Unambiguity

The notion of unambiguity is well-known with respect to automata and grammars. Usually, a nondeterministic automaton M is said to be *unambiguous* if for every input there is at most one accepting computation in M. Of course, every deterministic automaton is unambiguous. This leads, in a natural way, to the classes UnambSPACE(f) and UnambAuxPDA-TIME(f).

This concept is sometimes called *weak* unambiguity to distinguish it from the alternative notion of *strong unambiguity*, in which between any two configurations there is at most one computation path, which leads from the first configuration to the second one. Weak unambiguity demands this for pairs of initial and accepting configurations only. In this way we get the classes StUnambSPACE(f) and StUnambAuxPDA-TIME(f) [4]. By definition we have

$$\text{DSPACE}(f)\subseteq\text{StUnambSPACE}(f)\subseteq\text{UnambSpace}(f)\subseteq\text{NSPACE}(f)$$

and

$$\begin{aligned}\text{DAuxPDA-TIME}(f)&\subseteq\text{StUnambAuxPDA-TIME}(f)\\&\subseteq\text{UnambAuxPDA-TIME}(f)\\&\subseteq\text{NAuxPDA-TIME}(f).\end{aligned}$$

Considering the families CFL of context-free languages, DCFL of deterministic context-free languages, LIN of linear context-free languages, and DLIN[7] of languages accepted by deterministic pushdown automata, which do not push a symbol after a pop-move was performed, the following equations are well-known [22, 23]:

$$\begin{aligned}&\text{NAuxPDA-TIME(pol)}=\text{LOG(CFL)},\\&\text{DAuxPDA-TIME(pol)}=\text{LOG(DCFL)},\\(*)\quad&\text{NSPACE}(\log n)=\text{LOG(LIN)},\\&\text{DSPACE}(\log n)=\text{LOG(DLIN)}.\end{aligned}$$

Hence, one might be tempted to assume that

$$\begin{aligned}&\text{UnambAuxPDA-TIME(pol)}\stackrel{?}{=}\text{LOG(UnambCFL)},\\(**)\quad&\text{UnambSPACE}(\log n)\stackrel{?}{=}\text{LOG(UnambLIN)},\end{aligned}$$

where UnambCFL (UnambLIN) is the family of languages generated by unambiguous (linear) context-free grammars. Unfortunately, the corresponding constructions in (*) preserve determinism but not unambiguity.

If we want to transfer these concepts to boolean circuits, we have to exchange the notion of an accepting computation with that of an accepting subcircuit. In this way

[7] DLIN, which is easily seen to be DSPACE($\log n$)-complete even with respect to NC^1-reductions, is to be distinguished from the *deterministic linear languages* introduced in [12], which are contained in NC^1.

we would consider circuits which for all inputs have at most one accepting subcircuit, which is equivalent to the fact that no member of a certain, input-dependent set of "relevant" OR-gates is ever reached by more than one predecessor carrying a 1. A closer investigation shows that a more adequate notion is obtained by demanding this only for gates of unbounded fan-in and, in addition, by putting a dual restriction on AND-gates, too. Thus, we come to the following definition.

Definition 3.1. Let C be a circuit on n inputs built up by AND- and OR-gates of bounded fan-in and by $\exists$- and $\forall$-gates of unbounded fan-in. C is *unambiguous* if for all 2^n assignments of the inputs no $\exists$-gate receives a 1 by two or more of its predecessors and no $\forall$-gate receives a 0 by two or more predecessors. A circuit family $\mathscr{C} = \{C_n \mid n \geqslant 0\}$ is called *unambiguous* if C_n is unambiguous for each n.

This notion may be illustrated by the idea of a "vulnerable gate": We might think that all gates of unbounded fan-in in an unambiguous circuit are *vulnerable*, i.e. they may be destroyed if not handled properly. Thus, a vulnerable $\exists$-gate will correctly output a 0 if none of its inputs is equal to 1 and will output a 1 if exactly one of its inputs is a 1. But if there are more than one inputs carrying a 1 the gate is overloaded and, being vulnerable, will be destroyed. After that, we have no information concerning the output of this gate. Thus, in an unambiguous circuit all gates of unbounded fan-in are vulnerable and the unambiguity of the circuit implies that for no input-assignment any vulnerable gate is overloaded. Thus, we see that this notion of unambiguity corresponds to strong unambiguity. When constructing unambiguous circuits, terms like unambiguous disjunctions will be used to indicate that certain OR-gates will for no input receive more than one value 1 from their predecessors.

We mention in passing that in analogy to unambiguous context-free grammars, both the unambiguity of a circuit family and the exclusiveness (concerning read or write access) of a PRAM algorithm are undecidable.

Let us at this point stress the fundamental difference between the notions of unambiguity, i.e. demanding the uniqueness of certain computations, and of uniqueness, i.e. excluding inputs which are accepted by more than one computation. While the former is a restriction and, thus, unambiguous classes are contained in the corresponding nondeterministic classes, the latter one is a tool; usually classes defined by uniqueness are incomparable with or containing the corresponding nondeterministic classes.

Using Definition 3.1 to define an unambiguous version of AC^k, $k \geqslant 0$, leads straightforwardly to the family $\mathrm{UnambAC}^k$. For SAC^k, however, we have two different possibilities which lead to the families $\mathrm{UnambSAC}^k$ and $\mathrm{UnambS^EAC}^k$.[8]

Definition 3.2. Let $\mathrm{UnambAC}^k$ be the family of all languages recognizable by log-space-uniform unambiguous circuit families of polynomial size and depth bounded by

[8] The superscript E stands for "extended". In [13, 14] $\mathrm{UnambSAC}^k$ was called $\mathrm{UnambRAC}^k$ and $\mathrm{UnambS^EAC}^k$ was called $\mathrm{UnambSAC}^k$. But, meanwhile, the former UnambRAC-families turned out to be the more appropriate unambiguous version of the SAC-families.

$O(\log^k n)$ using both AND-gates and OR-gates of bounded as well as of unbounded fan-in. UnambSAC^k denotes the class of all languages recognizable by the corresponding circuit families using only $\exists$-gates of unbounded fan-in and AND-gates of bounded fan-in. (Thus, in this case all disjunctions are "vulnerable".) Finally, UnambS^{E}AC^k consists of all languages recognizable by the corresponding circuit families using $\exists$-gates of unbounded fan-in, AND-gates of bounded fall-in and, in addition, OR-gates of bounded fan-in.

Thus, the difference between UnambSAC and UnambS^{E}AC-circuits is that in the latter ones we may use small OR-gates which are not vulnerable.

Proposition 3.3. $NC^k \subseteq UnambSAC^k$ *for* $k \geqslant 1$.

Proof. Let C be an NC^k-circuit. Without loss of generality we may assume C to be layered. Since we can modify an NC^k-circuit C without leaving NC^k in such a way that for every gate a in C there exists a gate $\bar{a}$ computing the complement of a, we can replace a robust OR of two gates a and b by the OR of $a \wedge b$, $\bar{a} \wedge b$, and $a \wedge \bar{b}$, which can never get a multiple input of 1's. This results in a layered UnambSAC^k-circuit (which does not contain gates of unbounded fan-in!). □

As a consequence, the UnambXAC^k-hierarchies intertwine with the NC-hierarchy:

$$NC^k \subseteq UnambSAC^k \subseteq UnambS^EAC^k \subseteq UnambAC^k \subseteq AC^k$$

$$\subseteq NC^{k+1},\ UnambS^EAC^k \subseteq SAC^k \subseteq AC^k.$$

SAC^k and UnambAC^k seem to be incomparable, since SAC^k coincides with NAuxPDA-TIME($c^{\log^k n}$) [24] and UnambAC^k will be shown to be equal to CREW-TIME($\log^k n$).

4. Results

In this section we investigate unambiguous circuits and, thereby, relate the concepts of unambiguity and of exclusiveness. Let us first consider the following relations known in the nondeterministic case:

$$NL \subseteq SAC^1 = NAuxPDA\text{-}TIME(pol).$$

We now show the following in the unambiguous case:

$$StUnambSPACE(\log n) \subseteq UnambSAC^1 \subseteq UnambAuxPDA(pol).$$

Proposition 4.1. $StUnambSPACE(\log n) \subseteq UnambSAC^1$.

Idea of proof. Given a logspace machine A with polynomial upper bound $p(n)$ of the running time, we let a uniformity machine B first compute $n := |v|$ and $T := p(n)$ for a given input v. B works with gates labelled $\langle K, K', t\rangle$ meaning that the configuration K of A reaches K' in *exactly* t steps. The output is a vulnerable disjunction of all $\langle K_0, K_f, t\rangle$, where $1 \leqslant t \leqslant T$, K_0 is the starting configuration, and K_f is a final configuration. The usual Savitch-decomposition of $\langle K, K'', t\rangle$ into $\langle K, K', \lceil t/2 \rceil\rangle$ and $\langle K', K'', \lfloor t/2 \rfloor\rangle$ is unambiguous since there can be at most one configuration K', which is reachable from K in exactly $\lceil t/2 \rceil$ steps and which reaches K'' in exactly $\lfloor t/2 \rfloor$ steps. Otherwise, A would not be strongly unambiguous. Thus, the corresponding gate labelled $\langle K, K'', t\rangle$ is a vulnerable disjunction (of unbounded fan-in) over all conjunctions of $\langle K, K', \lceil t/2 \rceil\rangle$ with $\langle K', K'', \lfloor t/2 \rfloor\rangle$. To make the construction layered, we assume $\langle K, K'', t\rangle$ to belong to the level $\lceil \log t \rceil$. If the levels of $\lceil t/2 \rceil$ and $\lfloor t/2 \rfloor$ are different, we have to insert appropriate delays between the conjunction belonging to K' and $\langle K', K'', \lfloor t/2 \rfloor\rangle$ for each K'. In addition, delays are needed between the $\langle K_0, K_f, t\rangle$-gates and the output $\exists$-gate. Obviously, the constructed circuit is DSPACE($\log n$)-uniform. (In fact, it is even DLOGTIME-uniform!) □

Remark 4.2. This construction does not work for unambiguous logspace Turing machines, since there might be ambiguous partial computations which are not part of any accepting computation but might lead to the destruction of a vulnerable gate. Meanwhile in [4] it was shown that StUnambSPACE($\log n$) is even contained in DAuxPDA-TIME(pol), which is a subset of UnambSAC^1 as will be shown in Theorem 4.9.

Remark 4.3. As a consequence of Proposition 4.1, we get DSPACE($\log n$)$\subseteq$ UnambSAC^1. Thus, DSPACE($\log n$)-uniformity seems to be an adequate notion when working with unambiguous circuits of unbounded or semi-unbounded fan-in.

Since it is possible in a similar way to build for each i UnambSAC^1-circuits C_i and C_i' which compute whether the ith bit of $f(v)$ is 1 or 0 for some logspace-computable function f, we get the following corollary.

Corollary 4.4. *For each* $k \geqslant 1$, UnambSAC^k, UnambS^{E}AC^k *and* UnambAC^k *are closed under* LOG-*reducibilities.*

Proposition 4.5. UnambSAC$^k \subseteq$ UnambAuxPDA-TIME($c^{\log^k n}$) *for* $k \geqslant 1$.

Idea of proof. We apply the corresponding algorithm for showing SAC$^k \subseteq$ NAPDA-TIME($c^{\log^k n}$). Working recursively (top-) down from the output gate to input gates of an UnambSAC^k-circuit we evaluate an AND-gate by two recursive calls to its predecessors. For a vulnerable $\exists$-gate we guess nondeterministically, but unambiguously, the uniquely existing predecessor carrying a 1. □

Observe that this algorithm does not work strongly unambiguously: in case of rejection there are several paths leading to the same (rejecting) configuration!

Remark 4.6. It seems to be at least difficult to extend Proposition 4.5 to the class $\text{UnambS}^{\text{E}}\text{AC}^1$. The trick to evaluate gates of bounded fan-in "deterministically" by working through both predecessors does not work for OR-gates: If an OR-gate has two vulnerable $\exists$-gates as predecessors, one of them carrying a 1, then the OR-gate will get a 1, too, regardless of which (computation) path we follow for the $\exists$-gate carrying a 0. Thus, our simulating machine would no longer work (weakly) unambiguously. This explanation shows that UnambAPDA-machines can simulate languages accepted by $\text{UnambS}^{\text{E}}\text{AC}$-circuits under the restriction that no OR-gate is a least common ancestor of any two vulnerable $\exists$-gates. (It could be remarked here that the technique of Proposition 3.3 to get rid of robust OR-gates is not applicable here, since we do not have negations in SAC-circuits.)

In the following we relate CREW-PRAMs and unambiguous AC-circuits.

Theorem 4.7. $\text{UnambAC}^k \subseteq \text{CREW-TIME}(\log^k n)$ *for* $k \geqslant 1$.

Idea of proof. As in the usual evaluation of uniform circuits by PRAMs, we first simulate the uniformity machine and associate with each gate a a cell of global memory *result*(a) and with each wire a processor. In addition, we use for each gate a a global memory cell *reached*(a) which is initially set to 1 for all input gates, and to 0 otherwise. This can be performed by a CREW-PRAM since the uniformity machine is logspace-bounded and $\text{DSPACE}(\log n) \subseteq \text{CREW-TIME}(\log n)$. Then, as in [21], we run through a loop of length equal to the depth of the simulated circuit. For the simulation of a gate a of bounded fan-in the process associated with the wire leading to the first son of a sequentially asks the *reached*-bits of all predecessors of a. As soon as they are set, it computes *result*(a) and sets *reached*(a) to 1.

The simulation of a $\exists$-gate a of unbounded fan-in is done in the following way: As soon as the predecessors of a are marked as reached (simultaneously, since the simulated circuit is layered!), the corresponding processors overwrite *result*(a) with a 1, if their predecessor carries a 1 in its result cell. Then the first son of a sets *reached*(a) to 1.

Since the simulated circuit is unambiguous, there can be at most one successful predecessor, which gives us a CREW-algorithm. $\forall$-gates are treated in a dual way. □

This leaves open the question for showing $\text{CREW-TIME}(\log^k n) \subseteq \text{UnambAC}^k$ in analogy to $\text{CRCW-TIME}(\log^k n) \subseteq \text{AC}^k$. To do so, it might seem necessary to do the AC^0-computations in [21] with vulnerable gates, which should be a hard task. But by making intensive use of the logarithmic bound of the word length, it will be possible to show the converse of Theorem 4.7. As a preparation, we begin by considering

CROW-PRAMs. First of all, we get as a consequence of Corollary 4.4 the following corollary.

Corollary 4.8. CROW-TIME(log n)$\subseteq$UnambAC^1.

Proof.

$$\begin{aligned}\text{CROW-TIME}(\log n) &\subseteq \text{DAuxPDA-TIME}(\text{pol}) \quad \text{(by [8])}\\ &\subseteq \text{LOG}(\text{DCFL}) \quad \text{(by [23])}\\ &\subseteq \text{LOG}(\text{UnambCFL})\\ &\subseteq \text{LOG}(\text{UnambAC}^1) \quad \text{(scc [20])}\\ &\subseteq \text{UnambAC}^1 \quad \text{(by Corollary 4.4).}\ \square\end{aligned}$$

We now strengthen this result by proving the following theorem.

Theorem 4.9. CROW-TIME(log^k n)$\subseteq$UnambSAC^k *for* $k\geqslant 1$.

Proof. The proof is based on the recognition of CROW-TIME(log n)-languages by deterministic auxiliary push-down automata in polynomial time due to Dymond and Ruzzo [8]. The simulation, following ideas of Fortune and Wyllie [9] or Goldschlager [10], begins with DAuxPDA-TIME(pol)-computable functions STATE, GLOBAL, and LOCAL, stating that

(i) STATE$(p,t)=j \Leftrightarrow$ processor p executes at step t program line j,

(ii) GlOBAL$(t,a)=b \Leftrightarrow$ the global memory cell a contains after step t the value b, and

(iii) LOCAL$(p,t,a)=b \Leftrightarrow$ the memory cell a of processor p after step t contains the value b.

Here $1\leqslant p\leqslant P$, where P is the number of processors, which is polynomial in n, $1\leqslant t\leqslant T$, where T is the running time, which is O(log n), and $1\leqslant j\leqslant K$, where K is the length of the program of each processor, which is O(1).

The values of these equations are determined recursively in the following way:

Recursion for STATE$(p,t)=\mu$:

We set STATE$(p,1)=1$ and for $t>1$ STATE$(p,t)=j \Leftrightarrow$

Unconditional jump:

($\exists_{1\leqslant\mu\leqslant K}$ statement S_μ is of the form "**goto** S_j": STATE$(p,t-1)=\mu$) **or**

Successful conditional jump (only for $t\geqslant 2$):

($\exists_{1\leqslant\mu\leqslant K}$ statement S_μ is of the form "**if** $L_a>0$ **then goto** S_j":

STATE$(p,t-1)=\mu$ **and** NOT LOCAL$(p,t-2,a)=0$ **or**

Unsuccessful conditional jump (only for $t \geqslant 2$):

(If statement S_{j-1} is of the form "**if** $L_a > 0$ **then goto** S_b":

STATE$(p, t-1) = j-1$ **and** LOCAL$(p, t-2, a) = 0$) **or**

Otherwise:

(If statement S_{j-1} is not a jump: STATE$(p, t-1) = j-1$).

Recursion for GLOBAL$(t, i) = j$:

Let p be the (index of the) write-owner of global memory cell G_i. p is computable in logarithmic space according to our model of a CROW-PRAM and, thus, can be found out by the uniformity machine of the circuit to be constructed. We set GLOBAL$(0, i) = j$ if j is the ith bit of the input for $i \leqslant n$, GLOBAL$(0, i) = 0$ for $i > n$, and for $t \geqslant 1$ we have GLOBAL$(t, i) = j \Leftrightarrow$

The content of G_i was rewritten in step t:

($\exists_{1 \leqslant \mu \leqslant K}$ statement S_μ is of the form "$G_{L_a} = L_b$":

STATE$(p, t) = \mu$ **and** LOCAL$(p, t-1, a) = i$ **and** LOCAL$(p, t-1, b) = j$) **or**

p accessed the global memory without affecting G_i:

($\exists_{1 \leqslant \mu \leqslant K}$ statement S_μ is of the form "$G_{L_a} = L_b$":

STATE$(p, t) = \mu$ **and** NOT LOCAL$(p, t-1, a) = i$ **and**

GLOBAL$(t-1, i) = j$) **or**

Otherwise:

($\exists_{1 \leqslant \mu \leqslant K}$ statement S_μ is not of the form "$G_{L_a} = L_b$":

STATE$(p, t) = \mu$ **and** GLOBAL$(t-1, i) = j$).

Recursion for LOCAL$(p, t, i) = j$:

The definition of LOCAL(p, t, i) is a bit more extensive. By definition we know LOCAL$(p, 0, i) = 0$ and for $t \geqslant 1$ we have LOCAL$(p, t, i) = j \Leftrightarrow$

Read from global memory:

($\exists_{1 \leqslant \mu \leqslant K}$ statement S_μ is of the form "$L_i = G_{L_b}$":

STATE$(p, t) = \mu$ **and** ($\exists_{0 \leqslant a \leqslant q(n)}$ LOCAL$(p, t-1, b) = a$

and GLOBAL$(t-1, a) = j$)) **or**

Read from global memory without affecting L_i:

($\exists_{1 \leqslant \mu \leqslant K}$ statement S_μ is of the form "$L_a = G_{L_b}$", $a \neq i$:

STATE$(p, t) = \mu$ **and** LOCAL$(p, t-1, i) = j$) **or**

Indirect local write:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_{L_a} = L_b$":

STATE$(p,t) = \mu$ **and** LOCAL$(p,t-1,a) = i$ **and**

LOCAL$(p,t-1,b) = j$) **or**

Indirect local write without affecting L_i:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_{L_a} = L_b$":

STATE$(p,t) = \mu$ **and** NOT LOCAL$(p,t-1,a) = i$ **and**

LOCAL$(p,t-1,i) = j$) **or**

Indirect local read:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_i = L_{L_b}$":

STATE$(p,t) = \mu$ **and** ($\exists_{0 \leq a \leq q(n)}$ LOCAL$(p,t-1,b) = a$) **and**

LOCAL$(p,t-1,a) = j$)) **or**

Indirect local read without affecting L_i:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_a = L_{L_b}$", $a \neq i$:

STATE$(p,t) = \mu$ **and** LOCAL$(p,t-1,i) = j$) **or**

Binary operation:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_i = L_a \circ L_b$": STATE$(p,t) = \mu$ **and**

($\exists_{0 \leq a' \leq q(n)}$ $\exists_{0 \leq b' \leq q(n)}$ $j = a' \circ b'$: LOCAL$(p,t-1,a) = a'$ **and**

LOCAL$(p,t-1,b) = b'$)) **or**

Binary operation not affecting L_i:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_c = L_a \circ L_b$", $c \neq i$:

STATE$(p,t) = \mu$ **and** LOCAL$(p,t-1,i) = j$) **or**

PIN, LENGTH *or constant assignment*:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_i =$ PIN" and $p = j$,

"$L_i =$ LENGTH" and $n = j$, or "$L_i = j$":

STATE$(p,t) = \mu$) **or**

PIN, LENGTH *or constant assignment not affecting* L_i:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is of the form "$L_a =$ PIN", "$L_a =$ LENGTH"

or "$L_a = j$", $a \neq i$:

STATE$(p,t) = \mu$ **and** LOCAL$(p,t-1,i) = j$) **or**

Otherwise:

($\exists_{1 \leq \mu \leq K}$ statement S_μ is not of the form "$L_a \cdots$" or "$L_{L_a} \cdots$":

STATE$(p,t)=\mu$ **and** LOCAL$(p,t-1,i)=j$).

In order to convert this structure into a circuit we will replace the predicates STATE$(p,t)=j$, GLOBAL$(t,i)=j$ and LOCAL$(p,t,i)=j$ by gates labelled $\langle$STATE$(p,t),j\rangle$, $\langle$GLOBAL$(t,i),j\rangle$ and $\langle$LOCAL$(p,t,i),j\rangle$. This is possible, since we assumed a logarithmic bound on the word length and, hence, have both an address space and a data space of polynomial size. Contrast this with the construction of Stockmeyer and Vishkin in [21], where data and addresses were coded bitwise. Their result implies a kind of normal form, stating that CRCW-PRAMs with a polynomial number of processors and a polynomial time bound can be transformed into equivalent machines with a logarithmic bound on the word length. But this construction depends heavily on the use of the concurrent write feature.

According to the recursion structures of STATE$(p,t)=j$, GLOBAL$(t,i)=j$ and LOCAL$(p,t,i)=j$, the corresponding circuits consist of negations, conjunctions of bounded fan-in, and disjunctions, some of which are of unbounded fan-in. In this way, it is obvious to translate these recursions into a layered and DSPACE(log n)-uniform AC^k-circuit. It is to be observed that the outermost disjunctions are not only bounded (by K) but are unambiguous, too, since the statement a processor is executing at a certain time is uniquely determined. Furthermore, the unbounded disjunctions used in the formulation of binary operations are unambiguous, since for each pair (a',b') of possible values of LOCAL(p,t,a) and LOCAL(p,t,b) exactly one is "true", i.e. there is exactly one pair (a',b') fulfilling both LOCAL$(p,t,a)=a'$ and LOCAL$(p,t,b)=b'$.

Thus, in order to provide an UnambSAC^k-circuit the only remaining task is to get rid of negations. But this can be done, since all data and addresses are polynomially bounded; an expression NOT LOCAL$(p,t,i)=j$ is simply translated into a disjunction of all values unequal to j:

$$\text{NOT LOCAL}(p,t,i)=j \Leftrightarrow \exists_{0 \leq b < j} \text{LOCAL}(p,t,i)=b \text{ or}$$

$$\exists_{j < b \leq q(n)} \text{LOCAL}(p,t,i)=b.$$

This construction is unambiguous, since the content of any cell of memory is uniquely determined at any time. □

As a consequence, we obtain the following corollary.

Corollary 4.10. DCFL$\subseteq$UnambSAC^1.

Extending this construction, we come to the main result of this paper (Theorem 4.11).

Theorem 4.11. CREW-TIME(log$^k n$)$\subseteq$UnambAC^k *for* $k \geq 1$.

Proof. In order to extend the simulation of Theorem 4.9 to CREW-PRAMs we have to change the recursion structure of the equations for GLOBAL(t,i). The processor p writing on a global memory cell no longer can be computed in advance by the uniformity machine, but now has to be determined by the circuit. This is done by the following recursion: For $t \geqslant 1$ we have GLOBAL$(t,i)=j \Leftrightarrow$

The content of G_i has been modified in step t:

$(\exists_{1 \leqslant p \leqslant P(n)}\ \exists_{1 \leqslant \mu \leqslant K}$ statement S_μ is of the form "$G_{L_a}=L_b$":

STATE$(p,t)=\mu$ **and** LOCAL$(p,t-1,a)=i$ **and** LOCAL$(p,t-1,b)=j$) **or**

The content of G_i remained unchanged in step t:

(GLOBAL$(t-1,i)=j$ **and** NOT$(\exists_{1 \leqslant p \leqslant P(n)}\ \exists_{1 \leqslant \mu \leqslant K}$

statement S_μ is of the form "$G_{L_a}=L_b$":

STATE$(p,t)=\mu$ **and** LOCAL$(p,t-1,a)=i$)).

Replacing this construction for the corresponding part used in the proof of Theorem 4.9, we get a layered DSPACE(log n)-uniform AC^k-circuit. Since the simulated machine is a CREW-PRAM, we know there are no two processors p and p', $p \neq p'$, such that at any time t p executes a statement of the form "$G_{L_a}=\mathrm{L}_b$", p' executes a statement of the form "$G_{L_{a'}}=L_{b'}$", and LOCAL(p,t,a) coincides with LOCAL$(p't,a')$. But this implies that the disjunctions over $1 \leqslant p \leqslant P(n)$ are unambiguous, too. This proves our theorem. □

Corollary 4.12. CREW-TIME(log$^k n$)=UnambAC^k *for* $k \geqslant 1$.

Remark 4.13. Although the original method of [8] works on a DAuxPDA and helps us in simulating a CREW-PRAM by an unambiguous circuit, it does not seem possible to use it directly to show CREW-TIME(log n)$\subseteq$UnambAuxPDA-TIME(pol), since we use negations (or, equivalently, unambiguous $\forall$-gates) to treat the case that a global memory cell remains unchanged. Otherwise we would have CREW-TIME(log$^k n$)$\subseteq$SAC^k!

Rytter [20] was able to recognize unambiguous context-free languages with CREW-PRAMs in logarithmic time, which gives us Corollary 4.14.

Corollary 4.14. UnambCFL$\subseteq$UnambAC^1.

5. Discussion

Applying the concept of unambiguity, well-known from formal language theory, to circuits provided a new characterization of CREW-PRAMs.[9] More precisely, we may say that the relation of a CREW-PRAM to a CRCW-PRAM is comparable to that of

[9] Combining this new concept of an unambiguous circuit with a new type of gate called *select gate*, Niepel and Rossmanith [15] were able to give a circuit-based characterization of EREW-PRAMs.

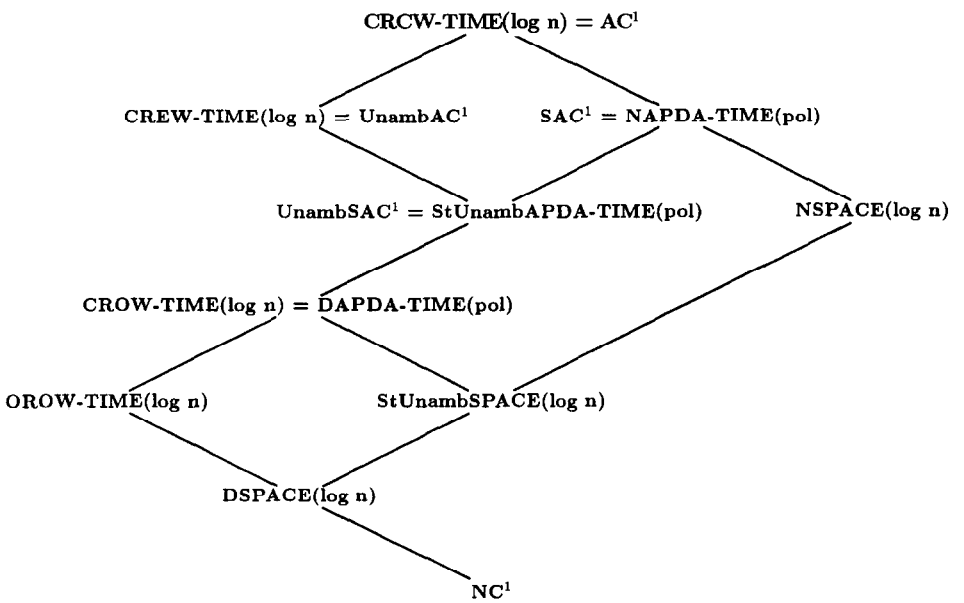

Fig. 1. The NC-structure between NC^1 and AC^1.

strong unambiguity to nondeterminism. This is why we introduced unambiguous circuits in a way corresponding to strong unambiguity. It is also possible to consider *weakly unambiguous circuits*, which as (strongly) unambiguous circuits are made of usual "robust" gates of bounded fan-in and of vulnerable gates of unbounded fan-in. But here we would allow vulnerable gates to be destroyed as long as this does not affect the result of the output. This means that the undefined output of a destroyed vulnerable gate would on all following paths finally be conjoined with the value FALSE or disjoined with the value TRUE. A corresponding "weak CREW-PRAM" model would allow the simultaneous write access to cells of global memory, which would be destroyed and made unreadable by that, but its final result must not be affected by this fact or by the content of the cell after its destruction. In [14] the equation $SAC^1 = NAuxPDA\text{-}TIME(pol)$ could be extended to both types of unambiguities by showing both $UnambSAC^k = StUnambAuxPDA\text{-}TIME(pol)$ and $Weak\,UnambSAC^k = UnambAuxPDA\text{-}TIME(pol)$. In particular, this yields $UnambCFL \subseteq UnambSAC^1$, thereby unifying and strengthening Corollaries 4.10 and 4.14.

Finally, the relationships between the classes considered so far are depicted in Fig. 1.

Acknowledgment

I thank Peter Rossmanith for many stimulating discussions. He and Inga Niepel pointed out the general form of Theorem 4.11 for arbitrary powers of the log-function.

Finally, I am indebted to the anonymous referees for their very careful, comprehensive, and helpful reports.

References

[1] J. Balcázar, J. Diáz and J. Gabárro, *Structural Complexity Theory I* (Springer, Berlin, 1988).
[2] J. Balcázar, J. Diáz and J. Gabárro, *Structural Complexity Theory II* (Springer, Berlin 1990).
[3] A. Borodin, On relating time and space to size and depth, *SIAM J. Comput.* **6**(4) (1977) 733–744.
[4] G. Buntrock, B. Jenner, K.-J. Lange and P. Rossmanith, Unambiguity and fewness for logarithmic space, in: *Proc. 8th Conf. on Fundamentals of Computation Theory*, Lecture Notes in Computer Science, Vol. 529 (Springer, Berlin, 1991) 168–178.
[5] A. Chandra, D. Kozen and L. Stockmeyer, Alternation, *J. ACM* **28** (1981) 114–133.
[6] S. Cook, Characterizations of pushdown machines in terms of time-bounded computers, *J. ACM* **18** (1971) 4–18.
[7] S. Cook, A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64** (1985) 2–22.
[8] P. Dymond and W. Ruzzo, Parallel RAMs with owned global memory and deterministic context-free language recognition, in: *Proc. 13th ICALP*, Lecture Notes in Computer Science, Vol. 226 (Springer, Berlin, 1986) 95–104.
[9] S. Fortune and J. Wyllie, Parallelism in random access machines, in: *Proc. 10th Ann. ACM Symp. on Theory of Computing* (1978) 114–118.
[10] L.M. Goldschlager, A unified approach to models of synchronous parallel computation, in: *Proc. 10th Ann. ACM Symp. on Theory of Computing* (978) 89–94.
[11] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Language, and Computation* (Addison-Wesley, Reading, MA, 1979).
[12] O.H. Ibarra, T. Jiang and B. Ravikumar, Some subclasses of context-free languages in NC^1, *Inform. Process. Lett.* **29** (1988) 11–117.
[13] K.-J. Lange, Unambiguity of circuits, In: *Proc. 5th IEEE Structure in Complexity Conf.* (1990) 130–137.
[14] K.-J. Lange and P. Rossmanith, Characterizing unambiguous augmented pushdown automata by circuits, in: *Proc. 15th MFCS*, Lecture Notes in Computer Science, Vol. 452 (Springer, Berlin, 1990) 399–406.
[15] I. Niepel and P. Rossmanith, Uniform circuits and exclusive read PRAMs, in: *Proc. 11th FST&TCS*, Lecture Notes in Computer Science, Vol. 560 (Springer, Berlin, 1991) 307–316.
[16] K.R. Reischuk, *Einführung in die Komplexitätstheorie* (Teubner, Stuttgart, 1990).
[17] P. Rossmanith, The owner concept for PRAMs, in: *Proc. 8th STACS*, Lecture Notes in Computer Science, Vol. 480 (Springer, Berlin, 1991) 172–183.
[18] W. Ruzzo, Tree-size bounded alternation, *J. Comput. System Sci.* **21** (1980) 218–235.
[19] W. Ruzzo, On uniform circuit complexity, *J. Comput. System Sci.* **22** (1981) 365–338.
[20] W. Rytter, Parallel time $O(\log n)$ recognition of unambiguous context-free languages, *Inform. and Control* **73** (1987) 75–86.
[21] L. Stockmeyer and C. Vishkin, Simulation of random access machines by circuits, *SIAM J. Comput.* **13** (1984) 409–422.
[22] I. Sudborough, A note on tape-bounded complexity classes and linear context-free languages, *J. ACM* **22** (1975) 499–500.
[23] I. Sudborough, On the tape complexity of deterministic context-free languages, *J. ACM* **25** (1978) 405–414.
[24] H. Venkateswaran, Properties that characterize LOGCFL, in: *Proc. 19th STOC* (1987) 141–150.
[25] K. Wagner and G. Wechsung, *Computational complexity* (VEB Deutscher Verlag der Wissenschaften, Berlin, 1986).
[26] R. Niedermeier and R. Rossmanith, Unambiguous simulations of auxiliary pushdown automata and circuits, in: *Proc. 1st Latin American Symp. on Theoretical Informatics*, Lecture Notes in Computer Science, Vol. 583 (Springer, Berlin, 1992) 389–400.